Software Engineering Institute

Ranged Integers for the C Programming Language

Jeff Gennari
Shaun Hedrick
Fred Long
Justin Pincar
Robert C. Seacord

September 2007

TECHNICAL NOTE
CMU/SEI-2007-TN-027

CERT Program

Carnegie Mellon

This work is sponsored by the U.S. Department of Defense.

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

Table of Contents

Abstract

This report describes an extension to the C programming language to introduce the notion of ranged integers, that is, integer types with a defined range of values. A variable of a ranged integer type will always have a value within the defined range as a result of initialization or assignment. Use of ranged integers would help prevent integer overflow errors and thus would result in more reliable and secure C programs. The syntax and semantics of ranged integers are presented, and some examples are given to illustrate their use.

1 Introduction

The inability of computers to represent an infinite range of values is well known. The behavior when a value is too large for an unsigned integer type to represent is defined as being "reduced modulo the number that is one greater than the largest value that can be represented by the resulting type" (or "wrapped around"—see ISO/IEC 9899:1999 TC2:2004 [ISO/IEC 2004a] Section 6.2.5.9). However, the behavior of a signed integer type when a value is too large or small to be represented is undefined and may result in modulo behavior or an exception (see ISO/IEC 9899:1999 TC2:2004 [ISO/IEC 2004a] Section 6.3.1.3).

In either the case of signed or unsigned integers, it is useful to define a valid range within which all values are guaranteed to lie after the result of an assignment or initialization on that integer type. It is then necessary to determine a policy to be enforced in the event that a resulting assignment or initialization lies outside of the valid range that is defined.

An extension to the C programming language's integer type system [ISO/IEC 2001] could provide such functionality through the use of *ranged integers*. This extension would effectively establish a new type of integer—a ranged integer—which maintains a specified policy on an assignment or initialization of the value of an integer. Ranged integers could be of particular value when used as array indices. The use of such indices in the C language is equivalent to unchecked pointer arithmetic and frequently results in reading and writing outside the bounds of an array, a condition frequently exploited as a buffer overflow vulnerability [Seacord 2005].

This report describes the semantics of ranged integer declaration and initialization and the policies enforced on operations in which they are used.[1]

1.1 TERMS AND DEFINITIONS

For the purposes of this description, the following definitions apply. Other terms are defined where they appear in the text and appear as italicized text. Terms explicitly defined are not to be presumed to refer implicitly to similar terms defined elsewhere. Terms not explicitly defined in this document are to be interpreted according to the C standard [ISO/IEC 2001].

range-min	The defined minimum value that a ranged integer is intended to hold.
range-max	The defined maximum value that a ranged integer is intended to hold.
storage policy	A set of rules to be enforced on an assignment or initialization into a ranged integer.
modwrap semantics	The ranged integer storage policy that enforces modulo behavior over the defined range of values.
saturation semantics	The ranged integer storage policy that stores range-max or range-min in the event of positive or negative overflow (respectively).

[1] This report is a development of an article by Robert C. Seacord, "Ranged Integers and Saturation Semantics" [Seacord 2007].

1.2 GOALS

Minimize the impact on the C language

One of the reasons that the C programming language has been so effective and popular is because of its ability to evolve but not generally break existing code. This proposal aims to adhere to this philosophy by making as little change to the standard as possible and to avoid defining notation, keywords, and so on that may break existing code. The extension to the C programming language to support embedded processors [ISO/IEC 2004b] introduced the reserved word `_Sat` to denote saturation semantics (of fixed point types). The syntax introduced in this report avoids the need for any new reserved words.

Minimize the performance overhead

The notion of dynamically checking that an integer type is within a certain range implies an associated runtime overhead, with both temporal and spatial implications. The approach described here allows implementations to define ranged integers in a way that minimizes the amount of space needed to store any associated data structures and aims for a performance overhead comparable to manually coded range checks.

Maximize the flexibility

If a ranged integer is not sufficiently flexible in how it can be used, it is not a viable alternative for hard-coded range checks and is consequently useless. The approach described here aims to make ranged integers sufficiently robust and flexible so that they become preferable to manual range checking in most circumstances.

2 Ranged Integers

Ranged integers are intended to be a security enhancement to the C programming language so that a stronger guarantee can be made about the value of an integer type after the results of an assignment or initialization.

As an example, consider an integer type that is used as an index into a fixed-length array. In many cases, this may result in a read or write out of bounds. However, if a malicious user is able to influence that value in some way, an exploitable buffer overflow might result. If a ranged integer were to be used instead, even if an unexpected value were attempted to be assigned to the integer type, a well-known policy would be in place to handle this condition and prevent the buffer overflow from occurring.

Ranged integers are intended to assist a developer by performing checks that would otherwise need to be done manually by control statements such as `if` and allow for a more intuitive policy to be enforced in the event of an exceptional condition.

2.1 DECLARATION

All of the C programming language's integer types—`char`, `short`, `int`, `long`, `long long` (both signed and unsigned; see ISO/IEC 9899:1999 TC2:2004 [ISO/IEC 2004a] Section 6.2.5)—and pointers or arrays thereof can be declared as ranged. The valid range for a given integer type is the implementation-defined minimum and maximum for that type. Range-min must not be arithmetically greater than range-max for any ranged integer.

A ranged integer may be declared with any of the C programming language storage class identifiers—`typedef`, `extern`, `static`, `auto`, or `register` (see ISO/IEC 9899:1999 TC2:2004 [ISO/IEC 2004a] Section 6.7.1). However, only ranged integers with the auto storage class can be declared as dynamic ranged integers.

A ranged integer may also be declared with any of the C programming language type qualifiers—`const`, `restrict`, or `volatile` (see ISO/IEC 9899:1999 TC2:2004 [ISO/IEC 2004a] Section 6.7.3). However, it should be noted that declaring a ranged integer as `const` or `volatile` may not have the intended effect. A `const` qualified ranged integer need not guarantee that any value stored into it after initialization be within its defined range, as assigning a value to a `const`-qualified integer after initialization is undefined. A `volatile`-qualified ranged integer may not necessarily always lie within the defined range as expected by definition of the `volatile` qualifier.

Pointers and arrays

As previously stated, both pointers to and arrays of ranged integers may be declared.

A ranged integer pointer points to either a ranged integer or NULL and must not point to a regular (non-ranged) integer. A regular (non-ranged) integer pointer must not point to a ranged integer.

An array of ranged integers degrades into a pointer to ranged integers when passed to another function. Although an array of ranged integers is guaranteed to fill a contiguous block of memory, no guarantee is placed on the layout of the ranged integers stored within that array; that is, the values of the stored ranged integers need not occupy adjacent bytes in memory.

Syntax

integer-type

underlying-integer-type

ranged-integer-type

underlying-integer-type

`char, signed char, unsigned char`

`short, unsigned short`

`int, unsigned int`

`long, unsigned long`

`long long, unsigned long long`

ranged-integer-type

underlying-integer-type integer`<..>`*integer*	modwrap `semantics`
underlying-integer-type integer`|..|`*integer*	saturation `semantics`

Note that the ranged integer syntax is meant to be pictographical. The notation *min*<..>*max* indicates that the values can continue (wrap around) when the ends of the range are reached, whereas *min*|..|*max* indicates that the values are constrained at the ends of the range.

Static ranged integers

A static ranged integer is declared using only integer constant expressions. A static ranged integer must not have a range-min less than or a range-max greater than what can be represented by the underlying integer type.

Dynamic ranged integers

A dynamic ranged integer takes integer expressions determined at runtime for its range-min and range-max. The size of a dynamic ranged integer need not necessarily have the same size as a static ranged integer pointer.

2.2 INITIALIZATION

The rules for the value stored on the initialization of a ranged integer type are as specified in the section “As an lvalue” below. Note that if a ranged integer is left uninitialized, its initial value is indeterminate and need not be in the defined range.

2.3 RUNTIME CONSTRAINTS

In the event that the defined range is exceeded and a storage policy is applied, a *runtime constraint handler* may be called to take an appropriate action, and the default action of the runtime constraint handler is to do nothing. The existing mechanisms used by ISO/IEC TR 24731-1 [ISO/IEC 2006] are used to handle the runtime constraints. That is, the user may define a function that is to be called if an attempt is made to initialize or assign to a ranged integer outside of its defined range. The user can implement this function to print an error message or to do anything else that the user wishes. The function does not even have to return. For example, it could abort the program. If the function does return, the appropriate saturation or modwrap semantics are applied to the value that caused the constraint to determine the value to be stored in the ranged integer. The default constraint handler does nothing. It is unspecified whether a function call must actually take place if the default handler is used.

2.4 USAGE

It is important to understand that ranged integers are a new type of integer that implements a storage policy, that is, range checking and enforcement occur only upon assignment or initialization. The following subsections describe this in more detail.

As an rvalue

When a ranged integer appears in an expression as an rvalue, it is subjected to the C programming language's normal integer promotion rules and conversion ranks (see ISO/IEC 9899:1999:2001 Section 6.3.1.1 [ISO/IEC 2001]) for its declared integer type.

As an lvalue

When a ranged integer appears as an lvalue, the right-hand side of the expression is first evaluated according to the normal promotion rules and conversion ranks for integer types. If the value of the right-hand side is not within the range specified for the ranged integer, a runtime constraint handler is called, if one is defined. If the runtime constraint handler returns execution to where it was called, modwrap or saturation semantics are applied to the value before it is assigned to the ranged integer.

As a pointer or array

In almost all cases, a pointer or an array that is typed as a ranged integer can be treated as if it were a pointer or array to the underlying integer type. The main difference lies in pointer arithmetic, as a ranged integer type is not guaranteed to use the same amount of space as its underlying integer type.[2]

[2] For example, if "`int 0|..|15 *pri;`" is declared, adding one to `pri` does not necessarily add `sizeof(int)` to determine the address.

3 Examples

The declaration

```
int 0|..|20 index = 0;
```

declares the variable `index` to be a static ranged integer of underlying type `int` with a minimum value of 0, a maximum value of 20, and saturation semantics, initialized with the value 0. Any assignment to the variable `index` will ensure that its value remains in the range 0 to 20. The assignment

```
index = 25;
```

results in `index` having the value 20 (assuming that no constraint handler is in place that prevents `index` from being assigned a value.) When `index` is used as an rvalue, it is treated as if it were a normal variable of type `int`.

Assuming that the variables `min` and `max` are defined of some integer type and have values such that `min` is not greater than `max`, the declaration

```
long min<..>max circular;
```

declares the variable `circular` to be a dynamic ranged integer of underlying type `long` with a minimum value of `min`, a maximum value of `max`, and modwrap semantics. As it is not initialized, its initial value is indeterminate and could be any value, not necessarily one in the range `min` to `max`. The assignment

```
circular = max + 1;
```

results in `circular` having the value `min` (assuming that no constraint handler is in place that prevents `circular` from being assigned a value).

4 Conclusion

This report describes a syntax and semantics for adding ranged integers to the C programming language. Ranged integers provide guarantees about the value of an integer in the event of integer overflow. Their introduction and use would result in more reliable and secure C programs.

References

URLs are valid as of the publication date of this document.

[ISO/IEC 2001]
International Organization for Standardization (ISO) and International Electrotechnical Commission. ISO/IEC 9899:1999:2001, *Information Technology – Programming Languages, Their Environments and System Software Interfaces – Programming Language C.* Geneva, Switzerland: International Organization for Standardization, 2001. http://www.open-std.org/JTC1/SC22/WG14/

[ISO/IEC 2004a]
International Organization for Standardization (ISO) and International Electrotechnical Commission. ISO/IEC 9899:1999 TC2:2004, *Information Technology – Programming Languages, Their Environments and System Software Interfaces – Programming Language C – Technical Corrigendum 2.* Geneva, Switzerland: International Organization for Standardization, 2004. http://www.open-std.org/JTC1/SC22/WG14/

[ISO/IEC 2004b]
International Organization for Standardization (ISO) and International Electrotechnical Commission. ISO/IEC TR 18037:2004, *Information Technology – Programming Languages, Their Environments and System Software Interfaces – Extensions for the Programming Language C to Support Embedded Processors.* Geneva, Switzerland: International Organization for Standardization, 2004. http://www.open-std.org/JTC1/SC22/WG14/

[ISO/IEC 2006]
International Organization for Standardization (ISO) and International Electrotechnical Commission. ISO/IEC TR 24731-1:2006, *Information Technology – Programming languages, Their Environments and System Software Interfaces – Specification for Safer, More Secure C Library Functions.* Geneva, Switzerland: International Organization for Standardization, 2006. http://www.open-std.org/JTC1/SC22/WG14/

[Seacord 2005]
Seacord, Robert C. *Secure Coding in C and C++*. Boston, MA: Addison-Wesley Professional, 2005 (ISBN 0-321-33572-4).

[Seacord 2007]
Seacord, Robert C. “Ranged Integers and Saturation Semantics.” *The Art of Software Security Assessment* blog, January 18, 2007.
http://taossa.com/index.php/2007/01/18/ranged-integers-and-semantics/

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.		
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 2007	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Ranged Integers for the C Programming Language		5. FUNDING NUMBERS FA8721-05-C-0003
6. AUTHOR(S) Jeff Gennari, Shaun Hedrick, Fred Long, Justin Pincar, Robert C. Seacord		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2007-TN-027
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE
13. ABSTRACT (MAXIMUM 200 WORDS) This report describes an extension to the C programming language to introduce the notion of ranged integers, that is, integer types with a defined range of values. A variable of a ranged integer type will always have a value within the defined range as a result of initialization or assignment. Use of ranged integers would help prevent integer overflow errors and thus would result in more reliable and secure C programs. The syntax and semantics of ranged integers are presented, and some examples are given to illustrate their use.		
14. SUBJECT TERMS C programming language, information system security, secure programming		15. NUMBER OF PAGES 14
16. PRICE CODE		

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL